Expressiveness of full first-order constraints in the algebra of finite or infinite trees

Alain Colmerauer Thi-Bich-Hanh Dao

March 5, 2003

Abstract

We are interested in the expressiveness of constraints represented by general first order formulae, with equality as unique relation symbol and function symbols taken from an infinite set F. The chosen domain is the set of trees whose nodes, in possibly infinite number, are labelled by elements of F. The operation linked to each element f of F is the mapping $(a_1, \ldots, a_n) \mapsto b$, where b is the tree whose initial node is labelled f and whose sequence of daughters is $a_1, \ldots, a_n$.

We first consider tree constraints involving long alternated sequences of quantifiers $\exists\forall\exists\forall\ldots$. We show how to express winning positions of two-person games with such constraints and apply our results to two examples.

We then construct a family of strongly expressive tree constraints, inspired by a constructive proof of a complexity result by Pawel Mielniczuk. This family involves the huge number $\alpha(k)$, obtained by top down evaluating a power tower of 2's, of height k. By a tree constraint of size proportional to k, it is then possible to define a tree having exactly $\alpha(k)$ nodes or to express the multiplication table computed by a Prolog machine executing up to $\alpha(k)$ instructions.

By replacing the Prolog machine with a Turing machine we show the quasi-universality of tree constraints, that is to say, the ability to concisely describe trees which the most powerful machine will never have time to compute. We also rediscover the following result of Sergei Vorobyov: the complexity of an algorithm, deciding whether a tree constraint without free variables is true, cannot be bounded above by a function obtained from finite composition of simple functions including exponentiation.

Finally, taking advantage of the fact that we have at our disposal an algorithm for solving such constraints in all their generalities, we produce a set of benchmarks for separating feasible examples from purely speculative ones. Among others we notice that it is possible to solve a constraint of 5000 symbols involving 160 alternating quantifiers.

1 Introduction

The algebra of (possibly) infinite trees plays a fundamental role in computer science: it is a model for data structures, program schemes and program executions. As early as 1976, Gérard Huet proposed an algorithm for unifying infinite terms, that is solving equations in that algebra [12]. Bruno Courcelle has studied the properties of infinite trees in the scope of recursive program schemes [8, 9]. Alain Colmerauer has described the execution of Prolog II, III and IV programs in terms of solving equations and disequations in that algebra [4, 5, 6, 1]. Michael Maher has introduced and justified a complete theory of the algebra of infinite trees [13]. Among others, he has shown that in this theory, and thus in the algebra of infinite trees, any first order formula is equivalent to a Boolean combination of conjunctions of equations (partially or totally) existentially quantified. Sergei Vorobyov has shown that the complexity of an algorithm, deciding whether a formula without free variables is true in that theory, cannot be bounded above, by a function obtained from finite composition of simple functions, including exponentiation [16]. Pawel Mielniczuk [14] has shown a similar result in the theory of feature trees, but with a more constructive method, which has inspired a large part of the work presented here.

We have recently developed an algorithm for solving general first order constraints in the algebra of infinite trees [10, 11]. The purpose of this paper is not the presentation of this algorithm, but of examples, first imagined as tests, then extended to show the expressiveness of such general constraints. The paper is organized as follows:

We end this first section by making clear the notions of infinite trees algebra and first-order constraints in that algebra.

In the second section, we use two-partner games for defining constraints involving long sequences of quantifiers $\exists\forall\exists\forall\ldots$.

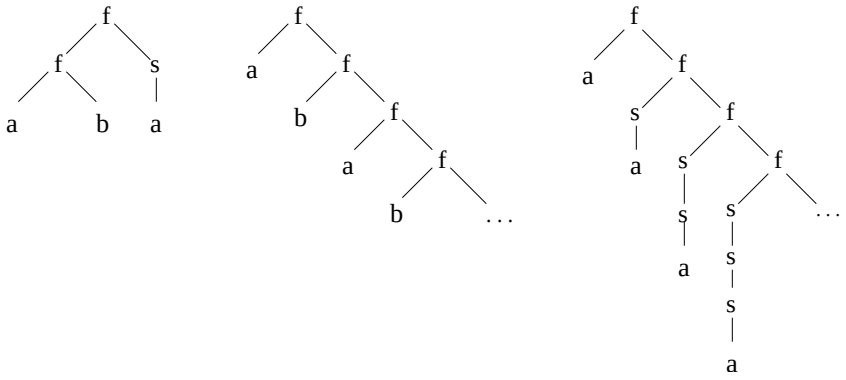
In the third section, we introduce a composition constraint which repeats the same constraint a tremendously large number of times. A long part of the section is devoted to proving its main property.

At section four, we move on to the most expressive constraints we know. They are obtained by changing the nature of the repeated constraint. We produce several examples, among which a constraint defining a huge finite tree and an almost perfect multiplication constraint. Then by simulating a Turing machine, we show the quasi-universality of tree constraint, that is to say, the ability to concisely describe trees which the most powerful machine will never have time to compute. This also allows us to give another proof of the complexity result of Sergei Vorobyov.

We conclude by discussions and benchmarks separating the feasible examples from the purely speculative ones.

1.1 The algebra of infinite trees

As usual, a *function symbol* is a symbol together with a non-negative integer, its *arity*. Trees, with nodes labelled by function symbols, are well known objects in the computer science world. Here are some of them:

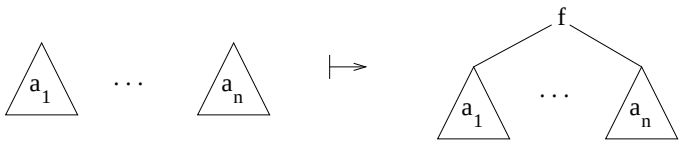


with, of course, the function symbols a, b, s, f having respectively the arities $0, 0, 1, 2$. Note that the first tree is the only one having a finite set of nodes, but that the second one has still a finite set of (patterns of) subtrees. We denote by $\mathbf{A}$ the set of all trees built on the infinite set F of function symbols.[1]

We equip $\mathbf{A}$ with a set of construction operations, one for each element $f \in F$, which are the mappings $(a_1, \ldots, a_n) \mapsto b$, where n is the arity of f and b the tree whose initial node is labelled

[1] More precisely we define first a *node* to be a word on the set of strictly positive integers. A *tree*, built on F, is then a mapping $a : E \to F$, where E is a non-empty set of nodes, each one $i_1 \ldots i_k$ (with $k \geq 0$) satisfying the two conditions: (1) if $k > 0$ then $i_1 \ldots i_{k-1} \in E$, (2) if the arity of $a(i_1 \ldots i_k)$ is n, then the set of nodes of E of the form $i_1 \ldots i_k i_{k+1}$ is obtained by giving to i_{k+1} the values $1, \ldots, n$.

f and whose sequence of daughters is $(a_1, \ldots, a_n)$:[2]



The set $\mathbf{A}$, together with the construction operations, is the *algebra of infinite trees* and is denoted by $(\mathbf{A}, F)$.

1.2 Tree constraints

We are interested in the expressiveness of constraints represented by general first order formulae, with equality as unique relation symbol and function symbols taken from an infinite set F. These *tree constraints* are of one of the 10 forms:

$$s = t,\ true,\ false,\ \neg(\varphi),\ (\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\varphi \to \psi),\ (\varphi \leftrightarrow \psi),\ \exists x\, \varphi,\ \forall x\, \varphi,$$

where φ and ψ are shorter tree constraints, x a variable taken from an infinite set and s, t terms, that is to say expressions of one of the forms:

$$x,\ \ f t_1 \ldots t_n,$$

where $n \geq 0$, $f \in F$, with arity n, and the t_i's are shorter terms.

The variables represent elements of the set $\mathbf{A}$ of trees built on F, the equality symbol is interpreted as equality and the function symbols f are interpreted as construction operations in the algebra of infinite trees $(\mathbf{A}, F)$. Thus a constraint without free variables is either true or false and a constraint $\varphi(x_1, \ldots, x_n)$ with n free variables x_i establishes an n-ary relation in the set of trees. For example the tree constraint

$$\psi(u, v, w, x, y) \stackrel{\text{def}}{=} \exists z \begin{bmatrix} z = f(x, f(y, z)) \wedge \\ z = f(u, f(v, f(w, z))) \end{bmatrix}$$

is equivalent to

$$x = u \wedge y = v \wedge x = w \wedge y = u \wedge x = v \wedge y = w.$$

Thus it expresses that the trees u, v, w, x, y are equal.

2 Long nesting of alternated quantifiers

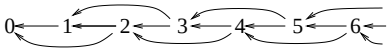
To show the expressive power of our constraints, we start examining deeply embedded quantifications. We consider two-partner games and characterize the positions from which it is always possible to win in at most k moves. In this manner we obtain constraints involving an alternated sequence of $2k$ quantifiers.

2.1 Winning positions in a two-partner game

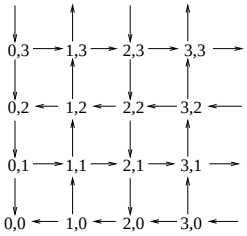
Let (V, E) be a directed graph, with V a set of vertices and $E \subseteq V \times V$ a set of edges. The sets V and E may be infinite and the elements of E are also called *positions*. We consider the following two-partner game: starting from an initial position x_0, each partner in turn, chooses a position x_1 such that $(x_0, x_1) \in E$, then a position x_2 such that $(x_1, x_2) \in E$, then a position x_3 such that

[2] In fact, the *construction* operation linked to the n-ary symbol f of F is the mapping $(a_1, \ldots, a_n) \mapsto b$, where the a_i's are any trees and b is the tree defined as follows from the a_i's and their sets of nodes E_i's: the set E of nodes of b is $\{\varepsilon\} \cup \{ix \mid x \in E_i \text{ and } i \in 1..n\}$ and, for each $x \in E$, if $x = \varepsilon$, then $b(x) = f$ and if x is of the form iy, with i being an integer, $b(x) = a_i(y)$.

$(x_2, x_3) \in E$ and so on... The first player who cannot make a move loses. For example the two following infinite graphs correspond to the two following games:



Game 1 A non-negative integer i is given and each partner, in turn, subtracts 1 or 2 from i, but keeping i non-negative. The first person who cannot play any more has lost.



Game 2 An ordered pair (i, j) of non-negative integers is given and each partner, in turn, chooses one of the integers i, j. Depending on whether the chosen integer u is odd or even, he then increases or decreases the other integer v by 1, but keeping v non-negative. The first person who cannot play any more has lost.

Let $x \in V$ be any vertex of the directed graph (V, E) and suppose that it is the turn of person A to play. The position x is said to be *k-winning* if, no matter the way the other person B plays, it is always possible for A to win, after having made at most k moves. The position x is said to be *k-losing* if, no matter the way A plays, B can always force A to lose and to play at most k moves.[3]

2.2 Expressing k-winning positions by a first-order constraint

Instead of tree constraints, consider general constraints, which are first-order formulae whose function and relation symbols are interpreted in a given structure $\mathcal{D}$, that is a set D together with operations and relations. Let $G = (V, E)$ be the graph representing a two-partner game, with $V \subseteq D$, and let $move(x, y)$ be a constraint in $\mathcal{D}$ such that:[4]

$$\text{for each } a \in V \text{ and } b \in V, \ \ move(a, b) \ \text{ iff } \ (a, b) \in E. \tag{1}$$

We want to build constraints $winning_k(x)$ and $losing_k(x)$ in $\mathcal{D}$, such that, for $k \geq 0$ and each $a \in V$,

$$\begin{array}{ll} winning_k(a) & \text{iff} \quad a \text{ is a } k\text{-winning position of } G, \\ losing_k(a) & \text{iff} \quad a \text{ is a } k\text{-losing position of } G. \end{array}$$

¿From the definition of k-winning and k-losing positions in the preceding section, we infer first that, for every $k \geq 0$:

$$\begin{array}{lcl} winning_0(x) & \leftrightarrow & false, \\ winning_{k+1}(x) & \leftrightarrow & \exists y \ move(x, y) \wedge losing_k(y), \end{array} \tag{2}$$

$$losing_k(x) \quad \leftrightarrow \quad \forall y \ move(x, y) \rightarrow winning_k(y). \tag{3}$$

[3] For the first game, it can be shown that the set of k-winning positions is the set of non-negative integers i such that $i < 3k$ and i is not divisible by 3. For the second game, it is the set of ordered pairs (i, j) of non-negative integers, such that $i+j < 2k$ and $i+j$ is odd.

[4] If $\varphi(x_1, \ldots, x_n)$ denotes a constraint whose free variables are among the x_i's, and if the a_i's are elements of D, then $\varphi(a_1, \ldots, a_n)$ is the interpretation of the constraint in $\mathcal{D}$, where the free occurrences of the x_i's are interpreted by the corresponding a_i's.

By using only existential quantifiers and unfolding the equivalences, we then notice that (2) can be replaced by

$$winning_k(x) \leftrightarrow \boxed{\begin{array}{l} \exists y\, move(x,y) \wedge \neg(\\ \exists x\, move(y,x) \wedge \neg(\\ \exists y\, move(x,y) \wedge \neg(\\ \exists x\, move(y,x) \wedge \neg(\\ \ldots \\ \exists y\, move(x,y) \wedge \neg(\\ \exists x\, move(y,x) \wedge \neg(\\ false \qquad \underbrace{)\ldots)}_{2k} \end{array}} \tag{4}$$

Thus equivalences (4) and (3) provide an explicit way for building the constraints we need. Notice that, by moving the negations down in (4), we get a nesting of $2k$ alternated quantifiers.[5]

It is possible to keep equivalence (4), while weakening condition (1). We first remark, that for any non-negative k, the following property holds:

Property 1 *Let three directed graphs be of the form $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G = (V_1 \cup V_2, E_1 \cup E_2)$. The graphs G_1 and G have the same set of k-winning positions, if both:*

1. *the sets of vertices V_1 and V_2 are disjoint,*
2. *for each $x \in V_2$, there exists $y \in V_2$ with $(x, y) \in E_2$.*

Indeed, from the first condition it follows that E_1 and E_2 are disjoint and thus that the set of k-winning positions of G is the union of the set of k-winning positions of G_1 with the set of k-winning positions of G_2. But this last set is empty because of the second condition.

It follows that:

Property 2 (Generalized *move*) *Equivalence* (4) *holds also for any constraint $move(x, y)$ obeying the three conditions:*

1. *for each $a \in V$ and $b \in V$, if $(a, b) \in E$ then $move(a, b)$,*
2. *for no $a \in D-V$ and no $b \in V$ we have $move(a, b)$,*
3. *for each $a \in D-V$, there exists $b \in D-V$ such that $move(a, b)$.*

2.3 Example : tree constraint for game 1

We now reconsider game 1 introduced in section 2.1. As structure $\mathcal{D}$ we take the algebra $(\mathbf{A}, F)$ of infinite trees constructed on a set F of function symbols including among others the symbols $0, s$, of respective arities $0, 1$. We code the vertices i of the game graph by the trees $s^i(0)$.[6] Let $G = (V, E)$ be the graph obtained this way.

Then, as generalized $move(x, y)$ constraint we can take:

$$\boxed{move(x,y) \stackrel{\text{def}}{=} x = s(y) \vee x = s(s(y)) \vee (\neg(x = 0) \wedge \neg(\exists u\, x = s(u)) \wedge x = y)}$$

and according to property 2, the set of k-winning positions of game 1 is the set of solutions in x of the constraint $winning_k(x)$ defined by (4).

[5] From the three equivalences in (2) and (3) it follows that $\forall x\, winning_k(x) \rightarrow winning_{k+1}(x)$ and $\forall x\, losing_k(x) \rightarrow losing_{k+1}(x)$, for any $k \geq 0$. Indeed, from the first and the last equivalence we conclude that the implications hold for $k = 0$ and, if we assume that they hold for a certain $k \geq 0$, from the last two equivalences we conclude that they also hold for $k+1$.

[6] Of course, $s^0(0) = 0$ and $s^{i+1}(0) = s(s^i(0))$.

For example, for $k=1$ the constraint $winning_k(x)$ is equivalent to

$$x=s(0) \vee x=s(s(0))$$

and, for $k=2$, to

$$x=s(0) \vee x=s(s(0)) \vee x=s(s(s(s(0)))) \vee x = s(s(s(s(0))))).$$

2.4 Example : tree constraint for game 2

We also reconsider game 2 introduced in section 2.1. As structure $\mathcal{D}$ we take the algebra $(\mathbf{A}, F)$ of infinite trees constructed on a set F of function symbols including among others the symbols $0, f, g, c$, of respective arities $0, 1, 1, 2$. We code the vertices (i,j) of the game graph by the trees $c(\overline{i},\overline{j})$ with $\overline{i} = (fg)^{\frac{i}{2}}(0)$, if i is even, and $\overline{i} = g(\overline{i-1})$, if i is odd.[7] Let $G=(V,E)$ be the graph obtained this way.

The perspicacious reader will convince himself that, as generalized constraint $move(x,y)$, we can take:

$$move(x,y) \stackrel{\text{def}}{=} transition(x,y) \vee (\neg(\exists u\, \exists v\, x=c(u,v)) \wedge x=y)$$

with

$$transition(x,y) \stackrel{\text{def}}{=} \begin{bmatrix} \exists u\, \exists v\, \exists w \\ \begin{bmatrix}(x=c(u,v) \wedge y=c(u,w)) \vee \\ (x=c(v,u) \wedge y=c(w,u))\end{bmatrix} \\ \wedge \\ \begin{bmatrix}(\exists i\, u=g(i) \wedge succ(v,w)) \vee \\ (\neg(\exists i\, u=g(i)) \wedge pred(v,w))\end{bmatrix}\end{bmatrix}$$

$$succ(v,w) \stackrel{\text{def}}{=} \begin{bmatrix}((\exists j\, v=g(j)) \wedge w=f(v)) \vee \\ (\neg(\exists j\, v=g(j)) \wedge w=g(v))\end{bmatrix}$$

$$pred(v,w) \stackrel{\text{def}}{=} \begin{bmatrix} (\exists j\, v=f(j) \wedge \begin{bmatrix}(\exists k\, j=g(k) \wedge w=j) \vee \\ (\neg(\exists k\, j=g(k)) \wedge w=v)\end{bmatrix}) \vee \\ (\exists j\, v=g(j) \wedge \begin{bmatrix}(\exists k\, j=g(k) \wedge w=v) \vee \\ (\neg(\exists k\, j=g(k)) \wedge w=j)\end{bmatrix}) \vee \\ (\neg(\exists j\, v=f(j)) \wedge \neg(\exists j\, v=g(j)) \wedge \neg(v=0) \wedge w=v)\end{bmatrix}$$

According to property 2, the set of k-winning positions of game 2 is the set of solutions in x of the constraint $winning_k(x)$ defined in (4).

For example, for $k=1$ the constraint $winning_k(x)$ is equivalent to

$$x=c(g(0),0) \vee x=c(0,g(0))$$

and, for $k=2$, to

$$\begin{bmatrix}x=c(0,g(0)) \vee x=c(g(0),0) \vee x=c(0,g(f(g(0)))) \vee \\ x=c(g(0),f(g(0))) \vee x=c(f(g(0)),g(0)) \vee x=c(g(f(g(0))),0)\end{bmatrix}$$

3 Composition of constraints

We now move on to a systematic way of compressing a conjunction of a very large number of constraints into a small constraint.

[7] Of course, $(fg)^0(x)=x$ and $(fg)^{i+1}(x)=f(g((fg)^i(x)))$.

3.1 Definition and properties

We introduce the integer $\alpha(k)$, defined for each integer $k \geq 0$, by

$$\alpha(0) = 1, \qquad \alpha(k+1) = 2^{\alpha(k)}.$$

Thus

$$\alpha(n) = \underbrace{2^{\cdots^{\left(2^{\left(2^2\right)}\right)}}}_{n}.$$

The function α increases in a stunning way, since $\alpha(0) = 1$, $\alpha(1) = 2$, $\alpha(2) = 4$, $\alpha(3) = 16$, $\alpha(4) = 65536$ and $\alpha(5) = 2^{65536}$. Thus $\alpha(5)$ is greater than 10^{20000}, a number probably much greater than the number of atoms of the universe and the number of nanoseconds which elapsed since its creation!

We agree that the size $|\varphi|$ of a constraint φ, is the number of occurrences of all symbols, except parentheses and commas. (Constraints can be written in infix notation.) We denote by $||\varphi||$ the number of different symbols occurring in φ, except parentheses and commas. If x, y are variables and $\varphi(x, y)$ is a constraint then, for each $n \geq 0$, the *n-fold composition* of $\varphi(x, y)$ is the constraint denoted and defined by:[8]

$$\varphi^n(x, y) \stackrel{\text{def}}{=} \exists u_0 \ldots \exists u_n \; x = u_0 \wedge \varphi(u_0, u_1) \wedge \varphi(u_1, u_2) \wedge \cdots \wedge \varphi(u_{n-1}, u_n) \wedge u_n = y \tag{5}$$

We assume that the set $\mathbf{A}$ of trees is constructed over a set F of function symbols including among others the symbols $1, 2, 3$, of arity zero, and the symbol f, of arity four. Given a constraint $\varphi(x, y)$, for each $k \geq 0$ we introduce the constraint:

$$\boxed{\mathit{supercomposition}_k[\varphi](x, y) \stackrel{\text{def}}{=} \exists z \; \mathit{triangle}_k(3, x, z, y)} \tag{6}$$

with,

$$\boxed{\begin{array}{lcl} \mathit{triangle}_0(t, x, z, y) & \stackrel{\text{def}}{=} & z = x \wedge z = y \\ \mathit{triangle}_{k+1}(t, x, z, y) & \stackrel{\text{def}}{=} & \left[\begin{array}{l} [\exists u_1 \, \exists u_2 \; z = f(x, u_1, u_2, y)] \\ \wedge \\ \left[\begin{array}{l} \forall t' \, \forall y' \, \forall z' \\ \left[\begin{array}{l} (t' = 1 \vee t' = 2) \wedge \\ \mathit{triangle}_k(t', z, y', z') \end{array}\right] \rightarrow \\ \left[\begin{array}{l} (t' = 1 \wedge \mathit{form1}(z')) \vee \\ \left(t' = 2 \wedge \left[\begin{array}{l} \exists u \, \exists v \; \mathit{form2}(u, z', v) \wedge \\ (t = 1 \rightarrow \mathit{son}(u, v)) \wedge \\ (t = 2 \rightarrow \mathit{son}(u, v) \vee u = v) \wedge \\ (t = 3 \rightarrow \varphi(u, v)) \end{array}\right]\right) \end{array}\right] \end{array}\right] \end{array}\right] \end{array}} \tag{7}$$

and

$$\boxed{\begin{array}{lcl} \mathit{form1}(x) & \stackrel{\text{def}}{=} & \exists u_1 \ldots \exists u_4 \; x = f(u_1, f(u_2, u_2, u_2, u_2), f(u_3, u_3, u_3, u_3), u_4) \\ \mathit{form2}(x, z, y) & \stackrel{\text{def}}{=} & \exists u_1 \ldots \exists u_6 \; z = f(u_1, f(u_1, u_2, u_3, x), f(y, u_4, u_5, u_6), u_6) \\ \mathit{son}(x, y) & \stackrel{\text{def}}{=} & \exists u_1 \ldots \exists u_4 \; x = f(u_1, u_2, u_3, u_4) \wedge (y = u_2 \vee y = u_3) \end{array}} \tag{8}$$

We notice first that the size of the constraint defined in (6) linearly depends on k. More precisely:

Property 3 $\boxed{|\mathit{supercomposition}_k[\varphi](x, y)| = 9 + k(155 + |\varphi(x, y)|).}$

[8] Of course, if $\varphi(x_1, \ldots, x_n)$ is a constraint, the x_i's variables and the t_i's terms, $\varphi(t_1, \ldots, t_n)$ denotes the same constraint with the t_i's substituted for all the free occurrences of the corresponding x_i's.

To show this property, it is sufficient to count:

$$\begin{array}{lcl}
|form1\,(z')| & = & 23, \\
|form2\,(u,z',v)| & = & 27, \\
|son\,(u,v)| & = & 23, \\
|triangle_{\,k+1}(t,x,z,y)| & = & 59 + |triangle_{\,k}(t,x,z,y)| + |form1\,(z')| + \\
 & & |form2\,(u,z',v)| + 2|son\,(u,v)| + |\varphi(u,v)|, \\
 & = & 155 + |triangle_{\,k}(t,x,z,y)| + |\varphi(u,v)|, \\
|triangle_{\,0}(t,x,z,y)| & = & 7, \\
|supercomposition_{\,k}[\varphi](x,y)| & = & 2 + |triangle_{\,k}(t,x,z,y)|,
\end{array}$$

and to conclude. This property is interesting only if one notices that:

Property 4 *It is possible to name the variables occurring in the family of constraints of the previous property, in such a way that, there exists an integer c, depending only on φ, with $||supercomposition_{\,k}[\varphi](x,y)|| \leq c$, for each $k \geq 0$.*

We remind the reader that $||\varphi||$ is the number of different symbols occurring in φ, except parentheses and commas.

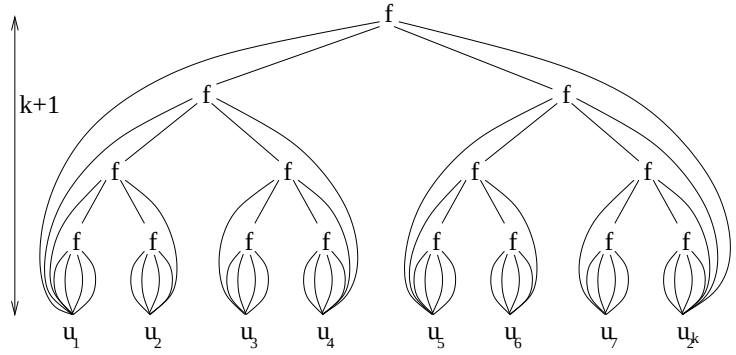
But the essential result is:

Theorem 1 $\boxed{supercomposition_{\,k}[\varphi](x,y) \;\leftrightarrow\; \varphi^{\,\alpha(k)-1}(x,y).}$

The next two subsections are devoted to the proof of the theorem.

3.2 Preliminaries to the proof of Theorem 1

We introduce trees, called *k-onions*, whose upper parts of depth k are essentially binary trees with additional branches permitting direct accesses to the nodes at depth $k+1$. For $k = 3$ such trees are of the form



(9)

where the u_i's are any trees. More generally, for $k \geq 0$, a *k-onion* is a tree z which satisfies the constraint $onion_{\,k}(z)$, with:

$$\begin{array}{lcl}
onion_{\,0}(z) & \stackrel{\text{def}}{=} & \exists u\; z = f(u,u,u,u) \\
onion_{\,k+1}(z) & \stackrel{\text{def}}{=} & \left[\begin{array}{l}
\exists u\, \exists u_1 \ldots \exists u_4\, \exists v\, \exists v_1 \ldots \exists v_4 \\
u = f(u_1,u_2,u_3,u_4) \wedge onion_{\,k}(u) \wedge \\
v = f(v_1,v_2,v_3,v_4) \wedge onion_{\,k}(v) \wedge \\
z = f(u_1,u,v,v_4)
\end{array}\right]
\end{array} \qquad (10)$$

The relevance of these trees comes from the property which follows. This property involves the constraints son, $form2$, defined by (8), and the notation $son^{\,i}$, defined by (5).

Property 5 *For every tree constraint $\varphi(x,y)$ and every $k \geq 0$, the following equivalence holds*[9]*:*

$$\varphi^{2^k-1}(x,y) \;\leftrightarrow\; \exists z \left[\begin{array}{l} onion_k(z) \\ \wedge \\ [\exists v_2 \exists v_3\; z = f(x,v_2,v_3,y)] \\ \wedge \\ \left[\begin{array}{l} \forall z' \\ \left[\bigvee_{i=0}^{k-1} son^i(z,z')\right] \rightarrow \\ \left[\begin{array}{l} \exists u \exists v\; form2(u,z',v) \wedge \\ \varphi(u,v) \end{array}\right] \end{array}\right] \end{array}\right]$$

PROOF We proceed by induction on k. We first show that the property holds for $k=0$. In that case, $\bigvee_{i=0}^{k-1} son^i(z,z') = false$ and the implication occurring in the right member of the equivalence is equivalent to *true*. Thus it only remains to prove that

$$x = y \;\leftrightarrow\; \exists z\; onion_0(z) \wedge [\exists v_2\; \exists v_3\; z = f(x,v_2,v_3,y)],$$

which is a direct consequence of the definition of $onion_0(z)$. Let us now assume that the property holds for $k \geq 0$ and let us show that it holds for $k+1$. Thus, under this assumption, we must show that the constraint $\varphi^{2^{k+1}-1}(x,y)$ is equivalent to

$$\exists z \left[\begin{array}{l} onion_{k+1}(z) \wedge [\exists w_1 \exists w_2\; z = f(x,w_1,w_2,y)] \wedge \\ \bigwedge_{i=0}^{k} [\forall z' son^i(z,z') \rightarrow [\exists u \exists v\; form2(u,z',v) \wedge \varphi(u,v)]] \end{array}\right].$$

Taking into account the definition of $onion_{k+1}(z)$ and splitting the case where $i=0$ from the cases where $i>0$, we get the equivalent constraint

$$\exists z \left[\begin{array}{l} \left[\begin{array}{l} \exists z_1\; \exists u_1 \ldots \exists u_4\; \exists z_2\; \exists v_1 \ldots \exists v_4 \\ z_1 = f(u_1,u_2,u_3,u_4) \wedge onion_k(z_1) \wedge \\ z_2 = f(v_1,v_2,v_3,v_4) \wedge onion_k(z_2) \wedge \\ z = f(u_1,z_1,z_2,v_4) \end{array}\right] \wedge [\exists w_1 \exists w_2\; z = f(x,w_1,w_2,y)] \wedge \\ [\exists u \exists v\; form2(u,z,v) \wedge \varphi(u,v)] \wedge \bigwedge_{i=1}^{k} [\forall z' son^i(z,z') \rightarrow [\exists u \exists v\; form2(u,z',v) \wedge \varphi(u,v)]] \end{array}\right],$$

which simplifies into

$$\exists z\; \exists z_1\; \exists u_2 \ldots \exists u_4\; \exists z_2\; \exists v_1 \ldots \exists v_3 \left[\begin{array}{l} z_1 = f(x,u_2,u_3,u_4) \wedge onion_k(z_1) \wedge \\ z_2 = f(v_1,v_2,v_3,y) \wedge onion_k(z_2) \wedge \\ z = f(x,z_1,z_2,y) \wedge \varphi(u_4,v_1) \wedge \\ \bigwedge_{i=1}^{k} [\forall z' son^i(z,z') \rightarrow [\exists u \exists v\; form2(u,z',v) \wedge \varphi(u,v)]] \end{array}\right]$$

and thus is equivalent to

$$\exists u_4\; \exists v_1 \left[\begin{array}{l} \exists z_1\; onion_k(z_1) \wedge \\ [\exists u_2 \exists u_3\; z_1 = f(x,u_2,u_3,u_4)] \wedge \\ \left[\begin{array}{l} \forall z' \\ \left[\bigvee_{i=0}^{k-1} son^i(z_1,z')\right] \rightarrow \\ \left[\begin{array}{l} \exists u \exists v\; form2(u,z',v) \wedge \\ \varphi(u,v) \end{array}\right] \end{array}\right] \end{array}\right] \wedge \varphi(u_4,v_1) \wedge \left[\begin{array}{l} \exists z_2\; onion_k(z_2) \wedge \\ [\exists v_2 \exists v_3\; z_2 = f(v_1,v_2,v_3,y)] \wedge \\ \left[\begin{array}{l} \forall z' \\ \left[\bigvee_{i=0}^{k-1} son^i(z_2,z')\right] \rightarrow \\ \left[\begin{array}{l} \exists u \exists v\; form2(u,z',v) \wedge \\ \varphi(u,v) \end{array}\right] \end{array}\right] \end{array}\right].$$

Since we have assumed that Property 5 holds for k, the preceding constraint is equivalent to the constraint

$$\exists u_4\; \exists v_1\; \varphi^{2^k-1}(x,u_4) \wedge \varphi(u_4,v_1) \wedge \varphi^{2^k-1}(v_1,y),$$

[9] For $k=0$ we consider that $\bigvee_{i=0}^{k-1} son^i(z,z')$ is the logical constant *false*.

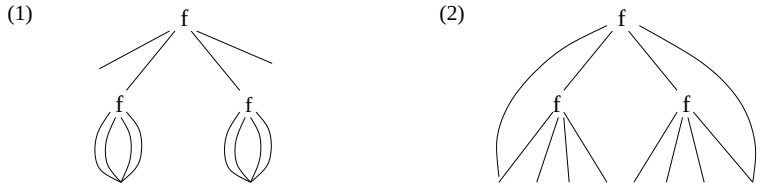
which is indeed equivalent to $\varphi^{2^{k+1}-1}(x,y)$. $\square$

For the next subsection we need an explicit definition of the constraint $onion_k(z)$. The property which follows does the job. It involves the constraints son, $form1$, $form2$, defined by (8), and the notation son^i, defined by (5).

Property 6 *For every tree z and every $k \geq 1$,*

$$onion_k(z) \;\leftrightarrow\; \left[\begin{array}{l} \left[\begin{array}{l}\forall z' \\ son^{k-1}(z,z') \rightarrow form1(z')\end{array}\right] \\ \wedge \\ \left[\begin{array}{l}\forall z' \\ \left[\bigvee_{i=0}^{k-1} son^i(z,z')\right] \rightarrow \left[\exists u \exists v\, form2(u,z',v)\right]\end{array}\right]\end{array}\right]$$

PROOF We proceed by induction on k. Property 6 is true for $k=1$. Indeed, according to the definition of $onion_1$ and $onion_0$, to be an 1-onion consists in being at the same time a tree of the form (1) and a tree of the form (2) below.



Thus

$$onion_1(z) \;\leftrightarrow\; form1(z) \wedge [\exists u \exists v\, form2(u,z,v)],$$

which is nothing else than Property 6, for $k=1$. Let us assume that Property 6 holds for k and let us show that it holds for $k+1$. Thus, under this assumption, we must show that the constraint $onion_{k+1}(z)$ is equivalent to the constraint

$$\left[\begin{array}{l}[\forall z'\, son^k(z,z') \rightarrow form1(z')] \wedge \\ \bigwedge_{i=0}^{k} [\forall z'\, son^i(z,z') \rightarrow [\exists u \exists v\, form2(u,z',v)]]\end{array}\right].$$

By splitting the case where $i=0$ from the cases where $i>0$, the preceding constraint can be written as the constraint

$$\left[\begin{array}{l}[\forall z'\, son^k(z,z') \rightarrow form1(z')] \wedge \\ [\exists u \exists v\, form2(u,z,v)] \wedge \\ \bigwedge_{i=1}^{k} [\forall z'\, son^i(z,z') \rightarrow [\exists u \exists v\, form2(u,z',v)]]\end{array}\right],$$

which is equivalent to the constraint

$$\left[\begin{array}{l}[\forall z'\, son^k(z,z') \rightarrow form1(z')] \wedge \\ \left[\begin{array}{l}\exists z_1 \exists u_1 \ldots \exists u_4 \exists z_2 \exists v_1 \ldots \exists v_4 \\ z_1 = f(u_1,u_2,u_3,u_4) \wedge z_2 = f(v_1,v_2,v_3,v_4) \wedge \\ z = f(u_1,z_1,z_2,v_4)\end{array}\right] \wedge \\ \bigwedge_{i=1}^{k} [\forall z'\, son^i(z,z') \rightarrow [\exists u \exists v\, form2(u,z',v)]]\end{array}\right],$$

which is equivalent to the constraint

$$\exists z_1 \exists u_1 \ldots \exists u_4 \exists z_2 \exists v_1 \ldots \exists v_4 \left[\begin{array}{l} z_1 = f(u_1,u_2,u_3,u_4) \wedge z_2 = f(v_1,v_2,v_3,v_4) \wedge \\ z = f(u_1,z_1,z_2,v_4) \\ [\forall z'\, son^{k-1}(z_1,z') \rightarrow form1(z')] \wedge \\ \bigwedge_{i=0}^{k-1} [\forall z'\, son^i(z_1,z') \rightarrow [\exists u \exists v\, form2(u,z',v)]] \wedge \\ [\forall z'\, son^{k-1}(z_2,z') \rightarrow form1(z')] \wedge \\ \bigwedge_{i=0}^{k-1} [\forall z'\, son^i(z_2,z') \rightarrow [\exists u \exists v\, form2(u,z',v)]]\end{array}\right],$$

which, given the assumption on k, is equivalent to the constraint

$$\exists z_1\,\exists u_1 \ldots \exists u_4\,\exists z_2\,\exists v_1 \ldots \exists v_4 \begin{bmatrix} z_1 = f(u_1,u_2,u_3,u_4) \wedge z_2 = f(v_1,v_2,v_3,v_4) \wedge \\ z = f(u_1,z_1,z_2,v_4) \\ onion_k(z_1) \wedge onion_k(z_2) \end{bmatrix},$$

which, by definition, is equivalent to $onion_{k+1}(z)$. $\square$

3.3 Proof of Theorem 1

We can now move on to the proof itself of Theorem 1. Given the definition of $supercomposition_k[\varphi](x,y)$, it is sufficient to show that, in the algebra of infinite trees, the last of the three following equivalences holds:

$$\begin{array}{lcl} (\exists z\ triangle_k(1,x,z,y)) & \leftrightarrow & son^{\alpha(k)-1}(x,y), \\ (\exists z\ triangle_k(2,x,z,y)) & \leftrightarrow & \bigvee_{i=0}^{i=\alpha(k)-1} son^i(x,y), \\ (\exists z\ triangle_k(3,x,z,y)) & \leftrightarrow & \varphi^{\alpha(k)-1}(x,y). \end{array} \tag{11}$$

By induction on k, we will prove that the three equivalences hold. They hold for $k=0$. Assuming that they hold for a given $k \geq 0$, let us prove that they hold for $k+1$.

By introducing an existential quantification on z, in both side of the definition of $triangle_{k+1}$ in (7) and by splitting up the $t'=1$ case from the $t'=2$ case, we get

$$\exists z\ triangle_{k+1}(t,x,z,y) \ \leftrightarrow\ \exists z \begin{bmatrix} [\exists u_1\,\exists u_2\, z = f(x,u_1,u_2,y)] \\ \wedge \\ \begin{bmatrix} \forall z' \\ (\exists y'\ triangle_k(1,z,y',z')) \rightarrow \\ form1\,(z') \end{bmatrix} \\ \wedge \\ \begin{bmatrix} \forall z' \\ (\exists y'\ triangle_k(2,z,y',z')) \rightarrow \\ \begin{bmatrix} \exists u\,\exists v\, form2\,(u,z',v) \wedge \\ \psi(u,v) \end{bmatrix} \end{bmatrix} \end{bmatrix},$$

with

$$\psi(u,v) \ \stackrel{\text{def}}{=}\ \begin{bmatrix} (t=1 \rightarrow son\,(u,v)) \wedge \\ (t=2 \rightarrow son\,(u,v) \vee u=v) \wedge \\ (t=3 \rightarrow \varphi(u,v)) \end{bmatrix}.$$

Since we have assumed that the three equivalences (11) hold for k, it follows that

$$\exists z\ triangle_{k+1}(t,x,z,y) \ \leftrightarrow\ \exists z \begin{bmatrix} [\exists u_2 \exists u_3\, z = f(x,u_2,u_3,y)] \\ \wedge \\ \begin{bmatrix} \forall z' \\ son^{\alpha(k)-1}(z,z') \rightarrow \\ form1\,(z') \end{bmatrix} \\ \wedge \\ \begin{bmatrix} \forall z' \\ [\bigvee_{i=0}^{\alpha(k)-1} son^i(z,z')] \rightarrow \\ \begin{bmatrix} \exists u\,\exists v\, form2\,(u,z',v) \wedge \\ \psi(u,v) \end{bmatrix} \end{bmatrix} \end{bmatrix},$$

that is to say, by looking back at Property 6,

$$\exists z\, triangle_{k+1}(t,x,z,y) \;\leftrightarrow\; \exists z \left[\begin{array}{l} onion_{\alpha(k)}(z) \\ \wedge \\ [\exists u_2 \exists u_3\, z = f(x,u_2,u_3,y)] \\ \wedge \\ \left[\begin{array}{l} \forall z' \\ \left[\bigvee_{i=0}^{\alpha(k)-1} son^i(z,z')\right] \rightarrow \\ \left[\begin{array}{l} \exists u \exists v\, form2\,(u,z',v) \wedge \\ \psi(u,v) \end{array}\right] \end{array}\right] \end{array}\right]. \qquad (12)$$

¿From (12), by using Property 5 with ψ instead of φ, we get

$$\exists z\, triangle_{k+1}(t,x,y,z) \;\leftrightarrow\; \psi^{\alpha(k+1)-1}(u,v)$$

and by successively assigning the values $1, 2, 3$ to t and taking into account the definition of ψ, we obtain (11).

4 Quasi-universality of tree constraints

As we will see, the composition constraint is a good starting point for defining expressive constraints.

4.1 Example: huge finite tree

First it is possible to define a huge finite tree, of $\alpha(k)$ nodes, with a small constraint, of size linearly depending on k. We suppose that the symbol 0, of arity zero, and the symbol s, of arity one, are elements of F and we introduce the constraint

$$\boxed{huge_k(x) \stackrel{\text{def}}{=} supercomposition_k[\varphi](x,0),}$$

with

$$\boxed{\varphi(x,y) \stackrel{\text{def}}{=} x = s(y).}$$

According to Property 3 and Theorem 1, we then have:

Property 7 $\boxed{|huge_k(x)| \;=\; 9{+}159k.}$

Property 8 $\boxed{huge_k(x) \;\leftrightarrow\; x = s^{\alpha(k)-1}(0).}$

4.2 Example : multiplication

Let us now try to express the multiplication constraint $w = uv$. Assume that the function symbols $0, s, empty, list, nat, plus, times$, of respective arities $0, 1, 0, 2, 1, 3, 3$, occur in F and consider the PROLOG like program,

$$\left\{\begin{array}{l} Times\,(0,v,0) \leftarrow Nat\,(v), \\ Times\,(s(u),v,w') \leftarrow Times\,(u,v,w) \wedge Plus\,(v,w,w'), \\ Plus\,(0,v,v) \leftarrow True, \\ Plus\,(s(u),v,s(w)) \leftarrow Plus\,(u,v,w), \\ Nat\,(0) \leftarrow True, \\ Nat\,(s(u)) \leftarrow Nat\,(u) \end{array}\right\},$$

which for a query of the form $Times\,(u, v, w)$ would enumerate the constraints of the form

$$u = s^m(0) \ \wedge \ v = s^n(0) \ \wedge \ w = s^{m \times n}(0), \tag{13}$$

ad infinitum, without forgetting any. For this program we construct the *Prolog machine* defined by the transition constraint

$$\varphi(x,y) \stackrel{\text{def}}{=} \boxed{\begin{array}{l} x = y \ \vee \\ \left[\begin{array}{l} \exists v \, \exists z \\ x = list(times\,(0, v, 0),\ z)) \wedge \\ y = list(nat\,(v),\ z) \end{array}\right] \ \vee \\ \left[\begin{array}{l} \exists u \, \exists v \, \exists w \, \exists w' \, \exists z \\ x = list(times\,(s(u), v, w'),\ z) \wedge \\ y = list(times\,(u, v, w),\ list(plus\,(v, w, w'),\ z)) \end{array}\right] \ \vee \\ \left[\begin{array}{l} \exists v \, \exists z \\ x = list(plus\,(0, v, v),\ z) \wedge \\ y = z \end{array}\right] \ \vee \\ \left[\begin{array}{l} \exists u \, \exists v \, \exists w \, \exists z \\ x = list(plus\,(s(u), v, s(w)),\ z) \wedge \\ y = list(plus\,(u, v, w),\ z) \end{array}\right] \ \vee \\ \left[\begin{array}{l} \exists z \\ x = list(nat\,(0),\ z) \wedge \\ y = z \end{array}\right] \ \vee \\ \left[\begin{array}{l} \exists u \, \exists z \\ x = list(nat\,(s(u)),\ z) \wedge \\ y = list(nat\,(u),\ z) \end{array}\right] \end{array}}$$

We see that, starting from configuration $list(nat\,(s^n(0)), z)$, the machine performs at most $n+1$ transitions and, starting from configuration $list(plus\,(s^n(0), v, v), z)$, the machine performs at most $n+1$ transitions. Thus, starting from configuration $list(times\,(s^m(0), s^n(0), w), z)$, the machine performs at most $(m+1)+(n+1)+m(n+1)$, that is $(m+1)(n+2)$ transitions. We conclude that the constraint

$$\exists x \, \exists y \, x = list(times\,(u, v, w), empty) \ \wedge \ \varphi^k(x, y) \ \wedge \ y = empty$$

is equivalent to the disjunction the constraints (13), with

$$(m+1)(n+2) \leq k.$$

¿From Property 3 and Theorem 1, it follows that the constraint

$$Times_k(u, v, w) \stackrel{\text{def}}{=} \boxed{\begin{array}{l} \exists x \, \exists y \\ x = list(times\,(u, v, w), empty) \wedge \\ supercomposition_k[\varphi](x, y) \wedge \\ y = empty \end{array}}$$

has the properties:

Property 9 $\boxed{|Times_k(u, v, w)| \ = \ 26 + 289k}$

Property 10 $\boxed{Times_k(u, v, w) \ \leftrightarrow \ \bigvee_{(m+1)(n+2) < \alpha(k)} (u = s^m(0) \ \wedge \ v = s^n(0) \ \wedge \ w = s^{m \times n}(0)).}$

By taking $k = 5$, we can then conclude:

With a tree constraint of 1471 symbols length, it is possible to express the multiplication table of integers ranging to the number of atoms of the universe.

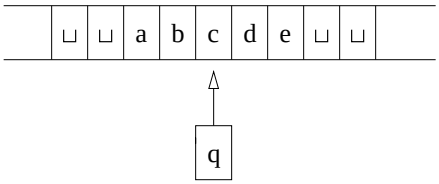
¿From this example we also deduce a systematic way to replace the "reasonable execution" of a PROLOG like program, by the process of solving a constraint of size comparable to the size of the program.

4.3 Quasi-universality

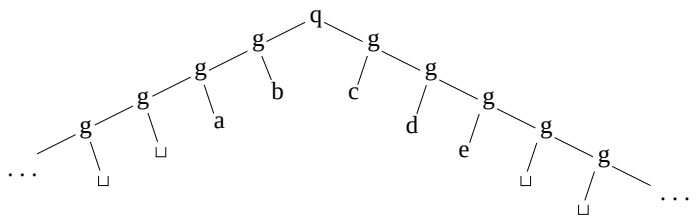
Instead of a Prolog machine we can take a Turing machine M, and express by $\varphi(x, y)$ its transition constraint, that is the fact that M may move from configuration x to configuration y by executing one instruction or the fact that $x = y$. As an example, we take the machine whose set of instructions is[10]

$$\left\{\begin{array}{ll}(q_0, 1, 1, q_1, R), & (q_0, \sqcup, \sqcup, q_2, L),\\ (q_1, 1, \sqcup, q_0, R), & (q_1, \sqcup, 1, q_2, L)\end{array}\right\} \tag{14}$$

whose states are q_0, q_1, q_2 and whose alphabet is $\{\sqcup, 1\}$. If we represent the machine configuration



by the infinite tree



then the transition constraint is

$$\varphi(x,y) \stackrel{\text{def}}{=} \boxed{\begin{array}{l} x = y \vee \\ \left[\begin{array}{l}\exists u\, \exists v\, \exists w \\ x = q_0(u, g(1, g(v, w))) \wedge \\ y = q_1(g(u, 1), g(v, w))\end{array}\right] \vee \left[\begin{array}{l}\exists u\, \exists v\, \exists w \\ x = q_0(g(u, v), g(\sqcup, w)) \wedge \\ y = q_2(u, g(v, g(\sqcup, w)))\end{array}\right] \vee \\ \left[\begin{array}{l}\exists u\, \exists v\, \exists w \\ x = q_1(u, g(1, g(v, w))) \wedge \\ y = q_0(g(u, \sqcup), g(v, w))\end{array}\right] \vee \left[\begin{array}{l}\exists u\, \exists v\, \exists w \\ x = q_1(g(u, v), g(\sqcup, w)) \wedge \\ y = q_2(u, g(v, g(1, w)))\end{array}\right] \end{array}}$$

where $\sqcup, 1, g, q_0, q_1, q_2$ are function symbols of respective arities $0, 0, 2, 2, 2, 2$.

Suppose, in general, that M is a Turing machine with alphabet $\Sigma \cup \{\sqcup\}$, initial state q_0, final state q_h and transition constraint $\varphi(x, y)$. For any word $a = a_1 a_2 \ldots a_m$ of $\Sigma^\star$, let $\overline{M}(a)$ be the

[10] Instruction (q, s, s', q', R) means: if the machine is in state q and its read-write head is in front of a cell containing symbol s, then it replaces s by s', moves its head one cell to the right and changes its state into q'. Instruction (q, s, s', q', L) means the same but with the head moving to the left.

element of $\Sigma^\star$ output by M on input a, if it exists.[11] Consider the tree constraints:

$$
\begin{array}{lcl}
\mathit{output}_k[\varphi,a](v) & \stackrel{\text{def}}{=} & \exists u \exists x \exists y\ \mathit{is}[a](u) \wedge \mathit{initial}(u,x) \wedge \mathit{supercomposition}_k[\varphi](x,y) \wedge \mathit{final}(y,v) \\
\mathit{is}[a](u) & \stackrel{\text{def}}{=} & \exists v\, u = g(a_1, g(a_2, \ldots, g(a_m, v)\ldots)) \wedge v = g(\sqcup, v), \\
\mathit{initial}(u,x) & \stackrel{\text{def}}{=} & \exists v\, x = q_0(v,u) \wedge v = g(v,\sqcup), \\
\mathit{final}(x,u) & \stackrel{\text{def}}{=} & \exists v\, x = q_h(v,u)
\end{array}
$$

¿From Property 3 and Theorem 1, we conclude that:

Property 11 $\boxed{|\mathit{output}_k[\varphi,a](v)| = 40 + 2|a| + k(155 + |\varphi(x,y)|).}$

Property 12 $\boxed{\mathit{output}_k[\varphi,a](v) \leftrightarrow \mathit{is}[\overline{M}(a)](v),}$ *under the restriction that the machine M executes fewer than $\alpha(k)$ instructions.*

We consider that a tree v which satisfies the constraint $\mathit{is}[b](v)$ is an *explicit coding* of the word $b = b_1 \ldots b_n$ and we denote by $|M|$ the number of instructions of the machine M. Then, from the two previous properties it follows that:

Corollary 13 (Quasi-universality of tree constraints) *For any words a, b and Turing machine M, such that M computes b from a in fewer than $\alpha(k)$ instructions, there exists a tree constraint $\psi(v)$ of size $\mathcal{O}(k + |M| + |a|)$, whose unique solution in v is a tree which codes explicitly b.*

4.4 Complexity

The tree constraints have a quasi-universal expressiveness in the sense of Corollary 13. Therefore the complexity of the algorithms for solving them must be very high. More precisely:

Theorem 2 *The time complexity of an algorithm, which decides whether a tree constraint without free variables is true, cannot be bounded above by an elementary function.*

Properties 11 and 12 will allow us to rediscover this result of Sergei Vorobyov [16] in the spirit of a proof of a similar result by Pawel Mielniczuk [14].

Let us first specify our terminology. An elementary function is a function obtained by finite compositions of the functions $x \mapsto \text{cst}$, $+$, $\times$, $(x,y) \mapsto x^y$. A Turing machine M is of *time complexity bounded above by $f(n)$* or just of *complexity $f(n)$*, if, for any word $a \in \Sigma^\star$, the machine executes at most $f(|a|)$ instructions for computing $\overline{M}(a)$, except eventually in a finite number of cases. A Turing machine M *decides the language* $L \subseteq \Sigma^\star$ if, for any $a \in \Sigma^\star$,

$$\overline{M}(a) = \begin{cases} 1 & \text{if } a \in L, \\ 0 & \text{if } a \notin L. \end{cases} \tag{15}$$

Finally, by $\langle\varphi\rangle$ we denote a word in $\Sigma^\star$ which codes in an obvious way the tree constraint φ.

By not distinguishing an algorithm from a Turing machine, theorem 2 can then be restated in more precise terms:

[11] At the beginning the machine is in the initial state q_0 and the twice infinite tape is filled with $\sqcup$ symbols, except a part, starting at the read-write head position, which contains the input $a \in \Sigma^\star$. Then the machine, which is supposed to be deterministic, executes the instructions until it reaches the final state q_h. The output is the longest element of $\Sigma^\star$ which starts at the final read-write head position. For example, with q_0 the initial state, q_2 the final state, with n 1's on input, machine (14) outputs the empty word, if n is even, and 11, if n is odd.

Theorem 2, restated *If N is a Turing machine such that, for all tree constraints ψ without free variables,*

$$\overline{N}(\langle\psi\rangle) = \begin{cases} 1 & \text{if } \psi \text{ is true,} \\ 0 & \text{if } \psi \text{ is false,} \end{cases} \tag{16}$$

then its time complexity cannot be bounded above by an elementary function.

PROOF Let's assume for the purpose of obtaining a contradiction that the restated theorem is false. Then there exists a Turing machine N, of elementary complexity, such that (16). Let L be a language, decidable by a Turing machine M of complexity $\alpha(n)$, but by no machine M' of elementary complexity. (For the existence of such a language, see for example the hierarchy theorems in the book of Michael Sipser [15]). Let $\varphi(x, y)$ be the transition constraint of M. It is then possible to build two Turing machines M' and A, such that, for any $a \in \Sigma^\star$,

$$\overline{M'}(a) = \overline{N}(\overline{A}(a)), \quad \overline{A}(a) = \langle \exists v \, output_{\,n}[\varphi, a](v) \wedge is\,[1](v)\rangle.$$

According to Property 11, we can impose that the complexity of machine A is elementary. Thus, N being of elementary complexity, M' is also of elementary complexity. But according to Property 12, M' decides the language L. This contradicts the definition of L and ends the proof. □

5 Discussions and conclusion

Thus, at the price of an incredible time complexity, it is possible to express anything by a tree constraint. How is it in practice?

Having at our disposal a general tree constraint solver, we have performed benchmarks on our four families of examples: expressing winning positions in two games, defining a huge finite tree and defining a multiplication table. The results are summarized in the following table, with CPU times given in milliseconds:

k	$winning_k$ game 1	$winning_k$ game 2	$huge_{\,k}$	$Times_{\,k}$
0	0	0	0	0
1	0	150	0	0
2	10	360	10	40
3	10	610	230	-
4	20	840	-	-
5	30	1180	-	-
10	300	5 970	-	-
20	4 270	236 350	-	-
40	89 870	-	-	-
80	3 841 220	-	-	-

The solver is programmed in C++ and the benchmarks are performed on a 350Mhz Pentium II processor, with 512Mb of RAM.

It must be noted that we were able to compute the k-winning positions of game 1, with $k = 80$, and of game 2, with $k = 20$. This corresponds respectively to a formula of 4961 symbols, involving 160 alternating quantifiers, and a formula of 7681 symbols involving 40 alternating quantifiers. We were prepared to experience difficulties in computing the tree of $\alpha(k)$ nodes, beyond $k = 3$, since $\alpha(4)$ is already 65536. With respect to multiplication, we were unable to succeed beyond $k = 2$ and had to satisfy ourselves with $0 \times 0 = 0$ and $0 \times 1 = 0$!

These tests have also removed some of our doubts about the correctness of the complicated formulae of our examples, even if, for readability, we have introduced predicates for naming subformulae. Of course the definitions of these predicates are supposed not to be circular and the solver unfolds and eliminates them in a first step. If circular definitions were accepted, then our constraints would look like generalized completions of logic programs [2].

Of all the described constraints, the Turing machine transition constraint is the only one using explicitly infinite trees. All the others are also meaningful in the algebra of finite trees. Since it is possible to simulate a Turing machine (in a more complicated way) in the algebra of finite trees, our results are also valid in this algebra. For other examples of constraints involving explicitly infinite trees, we refer the reader to [3, 7], where infinite trees are used for coding cyclic structures, like finite state automata, context-free grammars or λ-expressions.

Acknowledgement We thank Leszek Pacholski for our discussions during early summer 1998 in Marseille. We also thank the two anonymous reviewer for their detailed and helpful comments.

References

[1] Benhamou F., P. Bouvier, A. Colmerauer, H. Garetta, B. Giletta, J.L. Massat, G.A. Narboni, S. N'Dong, R. Pasero, J.F. Pique, Touraïvane, M. Van Caneghem and E. Vétillard, *Le manuel de Prolog IV*. PrologIA, Marseille, June 1996.

[2] Clark K.L., Negation as failure, in *Logic and Databases*, edited by H. Gallaire and J. Minker, Plenum Press, New York, pp. 293–322, 1978.

[3] Colmerauer A., Prolog and Infinite Trees, in *Logic Programming,* K.L. Clark and S.A.. Tarnlund editors, Academic Press, New York, pp. 231–251, 1982.

[4] Colmerauer A., Henry Kanoui and Michel Van Caneghem, Prolog, theoretical principles and current trends, in *Technology and Science of Informatics*, North Oxford Academic, vol. 2, no 4, August 1983. English version of the journal *TSI*, AFCET-Bordas, where the paper appears under the title: Prolog, bases théoriques et développements actuels.

[5] Colmerauer A., Equations and Inequations on Finite and Infinite Trees, in *Proceeding of the International Conference on Fifth Generation Computer Systems (FCGS-84)*, ICOT, Tokyo, pp. 85–99, 1984.

[6] Colmerauer A., An Introduction to Prolog III, *Communications of the ACM*, 33(7) : 68–90, 1990.

[7] Coupet-Grimal S. and O. Ridoux, On the use of advanced logic programming features in computational linguistics. *The Journal of Logic Programming*, 24(1&2), pages 121–159.

[8] Courcelle B., Fundamental Properties of Infinite Trees, *Theoretical Computer Science*, 25(2), pp. 95–169, March 1983.

[9] Courcelle B., Equivalences and Transformations of Regular Systems - Applications to Program Schemes and Grammars, *Theoretical Computer Science*, 42, pp. 1–122, 1986.

[10] Thi-Bich-Hanh Dao. Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis. In *Programmation en logique avec contraintes, JFPLC'2000*, pages 225–240. Hermès Sciences et Publication, 2000.

[11] Thi-Bich-Hanh Dao. *Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis*. PhD thesis, Université de la Méditerranée, December 2000.

[12] Huet G., *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$., Thèse d'Etat, Université Paris 7, 1976.

[13] Maher M.J., *Complete Axiomatization of the Algebra of Finite, Rational and Infinite Trees*, Technical report, IBM - T.J.Watson Research Center, 1988.

[14] Mielniczuk P., Basic Theory of Feature Trees, submitted to *Journal of Symbolic Computation*, available at `http://www.tcs.uni.wroc.pl/~mielni`.

[15] Sipser M., *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.

[16] Vorobyov S., An Improved Lower Bound for the Elementary Theories of Trees, *Proceeding of the 13th International Conference on Automated Deduction (CADE'96)*. Springer Lecture Notes in Artificial Intelligence, vol 1104, pp. 275-287, New Brunswick, NJ, July/August, 1996.